Naive Bayes

Naive because it assumes values are independant. In this set, we're doing it binomially since there are only two states for the target value.

In [1]:
```
import pandas
from sklearn.feature_extraction.text import TfidfVectorizer
import re
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.neural_network import MLPClassifier

testFract = 0.1 #fraction of cases used for the test, the rest is used for training

stressData = pandas.read_csv('Stress.csv', index_col='post_id')
stressData.head()
```

Out[1]:

post_id	subreddit	sentence_range	text	label	confidence	social_timestamp
8601tu	ptsd	(15, 20)	He said he had not felt that way before, sugge...	1	0.8	1521614353
8lbrx9	assistance	(0, 5)	Hey there r/assistance, Not sure if this is th...	0	1.0	1527009817
9ch1zh	ptsd	(15, 20)	My mom then hit me with the newspaper and it s...	1	0.8	1535935605
7rorpp	relationships	[5, 10]	until i met my new boyfriend, he is amazing, h...	1	0.6	1516429555
9p2gbc	survivorsofabuse	[0, 5]	October is Domestic Violence Awareness Month a...	1	0.8	1539809005

In [7]:
```
tfidf = TfidfVectorizer( binary=True) #No stopwords used because stopwords were ina

stressData['text'].replace('[\d][\d]+', ' # ', regex=True, inplace=True)
#only replacing numbers, not replacing caps or punctuation since caps and types of

x = stressData.text
y = stressData.label

xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=testFract, train_si

xTrain = tfidf.fit_transform(xTrain)
xTest = tfidf.transform(xTest)

naiveBayes = BernoulliNB()
naiveBayes.fit(xTrain, yTrain)
```

Out[7]:
```
BernoulliNB()
```

In [3]:
```
# make predictions on the test data
prediction = naiveBayes.predict(xTest)

print('accuracy = ', accuracy_score(yTest, prediction))
print('precision = ', precision_score(yTest, prediction))
print('recall = ', recall_score(yTest, prediction))
print('f1 = ', f1_score(yTest, prediction))
```

```
accuracy =  0.7464788732394366
precision =  0.6988636363636364
recall =  0.8661971830985915
f1 =  0.7735849056603774
```

Logisic Regression

Fucused on categorical classification instead of real number classification.

In [4]:
```
from sklearn.linear_model import LogisticRegression

#train
classifier = LogisticRegression(solver='lbfgs', class_weight='balanced')
classifier.fit(xTrain, yTrain)

predictionLR = classifier.predict(xTest)
print('accuracy = ', accuracy_score(yTest, predictionLR))
print('precision = ', precision_score(yTest, predictionLR))
print('recall = ', recall_score(yTest, predictionLR))
print('f1 = ', f1_score(yTest, predictionLR))
print('log loss =', log_loss(yTest, classifier.predict_proba(xTest)))
```

```
accuracy =  0.7112676056338029
precision =  0.6973684210526315
recall =  0.7464788732394366
f1 =  0.7210884353741497
log loss = 0.527923688834927
```

Neural Network

Four layers of "neurons" with an input, two hiiden, and an output layer. Hidden layers have been adjusted to fit as closely as needed.

In [5]:
```
x = tfidf.fit_transform(stressData.text)
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=testFract, train_si
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(6, 2), r
classifier.fit(xTrain, yTrain)
```

Out[5]:
```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(6, 2), random_state=1,
              solver='lbfgs')
```

In [6]:
```
predictionNN = classifier.predict(xTest)
print('accuracy = ', accuracy_score(yTest, predictionNN))
print('precision = ', precision_score(yTest, predictionNN))
print('recall = ', recall_score(yTest, predictionNN))
print('f1 = ', f1_score(yTest, predictionNN))
```

```
accuracy =  0.7147887323943662
precision =  0.7019867549668874
recall =  0.7464788732394366
f1 =  0.7235494880546076
```

Analysis

All classifiers use the tfidf Vectorizer in binary mode with stopwords included. Stopwords are included because their inclusion helped scientists easier detect depression and it seems to work that way as well with stress. Binary vectorization is used because it seems what words used when stressed are more important than how many. Both these options have been experimented with, and seem to produce better results. Naive Bayes did the best in accuracy and recall, but did poor in precision, which means examples classified as stressed aren't really stressed. This means more cases are falsely categorised as stressed. In comparison neural networks did mediocre, with the exception of it being the best in precision. This may because groups of words easier indicate wherther a sentence is not stressful than independent ones, which the naive bayes assumes. Logistic Regression suprisingly did worse than Naive bayes, which I wouldn't have expected since this is a more catergorical process. Maybe the fact that the ouput of Naive Bayes is a real number helps with accuracy since stress level is somewhat uncertain.